Tutorial

This tutorial provides a guided tour of some of the most commonly used debugging commands, with the opportunity to practice the following:

- *Invoke the source-level debugger (SLD) software, configure memory, and load a sample program.*
- *Navigate the source, memory, stack, and trace displays.*
- *Start and stop emulation in various ways.*
- *Collect and examine trace information.*

*The illustrations in this chapter show this tutorial running on various PowerPack EA and SW*Plus *emulators. You can use this tutorial with any PowerPack x86 emulator with overlay memory. Some displays differ between emulators and between processors.*

This tutorial assumes you have learned to use the Microsoft Windows environment. If you are unfamiliar with Windows, study your Windows tutorial before starting the SLD software.

Before starting this tutorial, connect the emulator to the SAST board.

This tutorial uses the \powerpak\samp386\demo.omf sample loadfile.

Configuring the SLD Software for the Tutorial

Use the PowerPack icon or your Start menu to start an emulator session. Note which (if any) buttons on your Toolbar are grayed-out to indicate unavailable operations.

The first time you start the debugger, you must select the COM port and baud rate for communication between your emulator and host system. The COM Port and Baud Rate dialogs appear automatically. Once you have selected communication settings, they are saved in your powerpak.ini file for future debugger invocations.

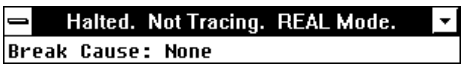
Use the Toolbar Map button to open the Map dialog. To map memory, use the Map dialog Add button and accept the default values in the Add dialog box.

Use the Toolbar Load button to load code and symbols from \powerpak\samp386\demo.omf.

Arrange Your Desktop

Simplify your desktop by minimizing or closing the Shell window. The Shell window icon is shown here in the left margin.

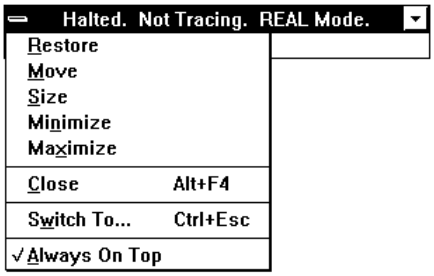
The first time you start the SLD software, the Status window also appears. This completes the initial default layout: Toolbar, Shell window, and Status window open and all other windows closed. The following shows the Status window before you have done any emulation or modified any memory or registers. The emulation processor is initially in real mode.



Minimize the Status window to simplify your desktop. The Status window icon is shown here in the left margin. Note the status message in the label below the icon.

According to the initial default layout, the Status window (whether open or iconized) remains visible regardless of any other SLD window position. To change this positioning, open the Status window Control menu and disable (toggle-off the checkmark) Always on Top.

Status window with open Control menu



Examine the Loaded Code and Symbols

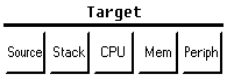
In this part of the tutorial, you will practice displaying different parts of the loaded code and symbolic information in the Source, Variable, and Memory windows. You will be changing only the window display and cursor position, without doing any emulation or changing the CS:EIP.

Display Source

From the Target section of the Toolbar, choose the Source button. You are viewing the startup module in the Source window. The associated source filename, startup.asm, appears in the title bar. The current CS:EIP (program counter) is marked by >>.

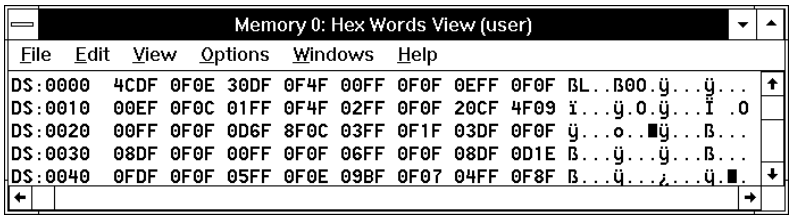
Source window showing the startup module

Display Memory



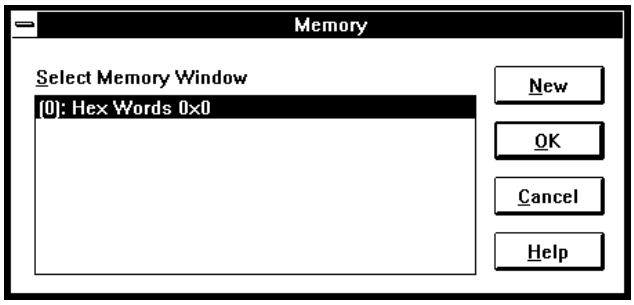
From the Target section of the Toolbar, choose the Mem button to display a Memory window. You are viewing the beginning of the data segment as hexadecimal word values.

Memory window showing the data segment



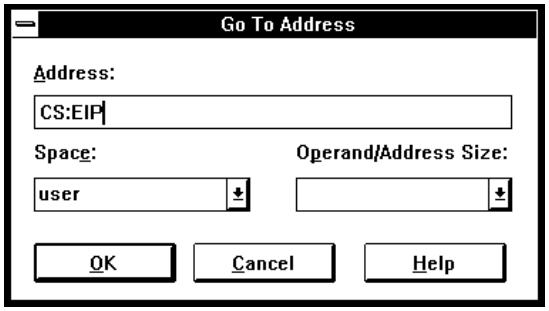
You can have up to 20 Memory windows open simultaneously. The first Memory window is labelled, in its title bar, Memory 0. Choose the Toolbar Mem button again to display the Memory dialog box for selecting or opening a Memory window. The following shows the Memory dialog box, with Memory window 0 available.

Memory dialog box for selecting an open or new Memory window

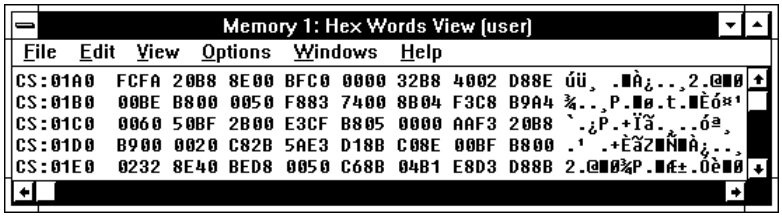


Choose the New button in the Memory dialog box. Another Memory window appears, labelled Memory 1. In this Memory window, open the Edit menu, choose Go To Address, and enter CS:EIP in the dialog box. The memory display changes to the program counter.

Go To Address dialog box for changing the Memory window display

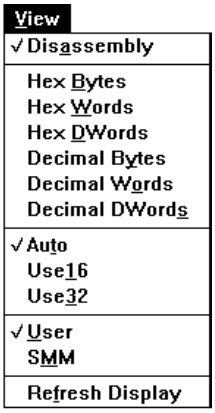


Memory window showing the code loaded at CS:EIP

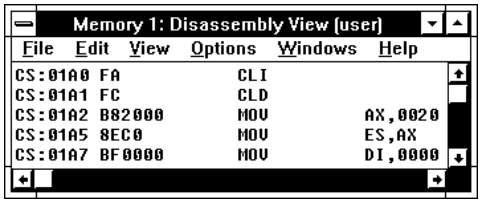


You can use multiple Memory windows to display different sections of memory or the same memory in different formats. Initially, both Memory window 0 and Memory window 1 display hexadecimal words (noted in their title bars). In Memory window 1, open the View menu and choose Disassembly.

Memory window View menu specifying disassembly

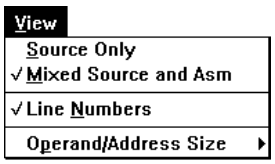


Memory window 1 showing memory contents as disassembly

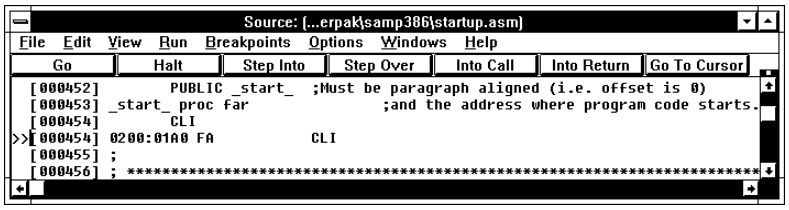


You can view the disassembly in the Source window corresponding to the disassembly in the Memory window. In the Source window, open the View menu and choose Mixed Source And Asm. The following shows the Source window View menu and the consequent display.

Source window View menu specifying mixed source and disassembly display

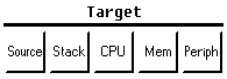


Source window showing disassembly interleaved with source



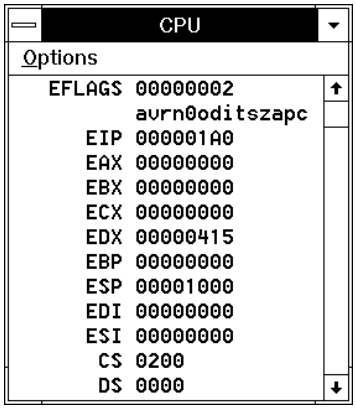
Examine the line of disassembly at the program counter, indicated by >> in the left margin. Compare the address and instruction to the Memory window 1 line showing address CS:01A0.

Display Registers



From the Target section of the Toolbar, choose the CPU button to display the CPU registers. The program counter consists of the CS and EIP registers. CS contains 0200 and EIP contains 01A0, also shown by the line with the >> program counter marker in the Source window in mixed view.

CPU window (EA-486 emulator)

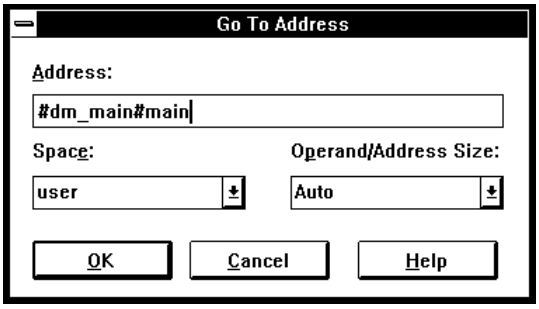


Find a Function by Symbolic Address

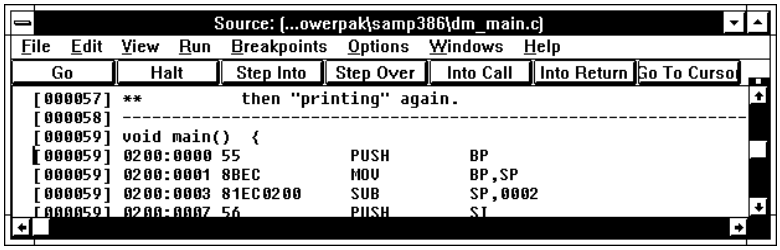
The section of code displayed when you first open the Source window is at the CS:EIP. Since no emulation has been done yet, the display is in the startup module. The title bar at the top of the Source window shows the source file for the displayed module.

Change the Source window display to the main function in the dm_main module, using the module and function symbols. In the Source window, open the Edit menu; choose Go To Address. Enter the fully-qualified symbol #dm_main#main in the Go To Address dialog box.

Go To Address dialog box to display the main function in the Source window



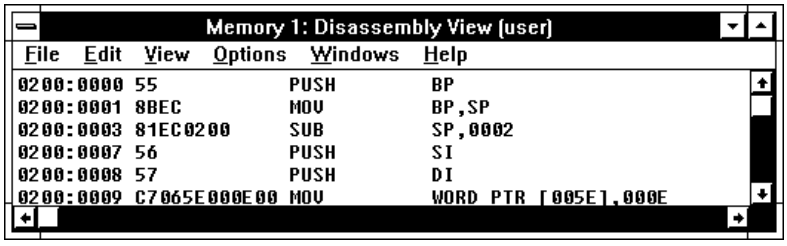
Source window showing main function in mixed source and assembly view



The emulator supports physical, linear, virtual, and symbolic addresses, interpreting numeric addresses as virtual unless you specify the L (linear) or P (physical) suffix.

In the Memory window, open the Edit menu; choose Go To Address. Enter #dm_main#main in the Go To Address dialog box, the same as you did for the Source window.

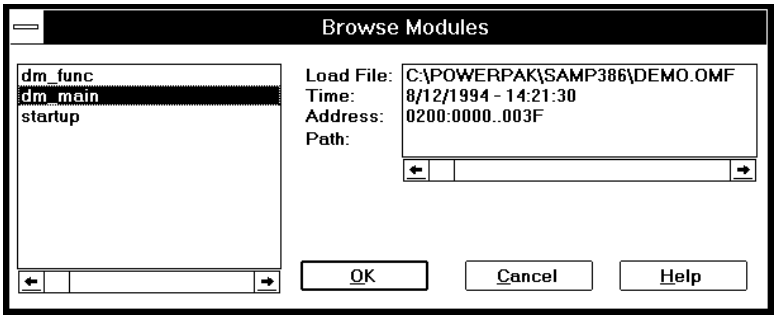
Memory window showing disassembly from the #dm_main#main address



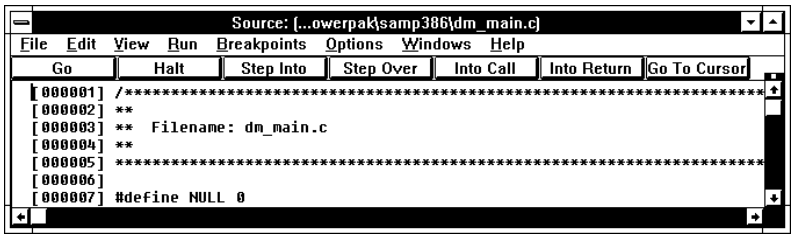
Find a Function By Module And Function Names

Display the dm_func module in the Source window. Open the File menu and choose Browse Modules. Select dm_func in the Browse Modules dialog box. Choose OK.

Browse Modules dialog box for displaying the dm_main module in the Source window

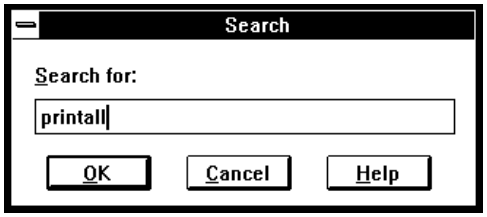


Source window displaying the beginning of the dm_main module in mixed source and assembly view.

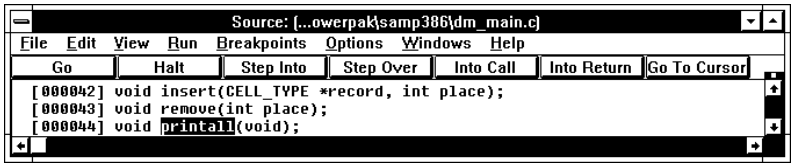


Find a reference to the printall function. Open the Edit menu and choose Search. Enter printall in the Search dialog box. The SLD software finds the first occurrance of printall.

Search dialog box for finding the first occurrance of the printall string in the Source window



Source window after a successful search for printall

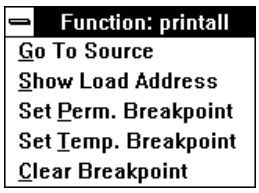


Find a Function From Any Reference

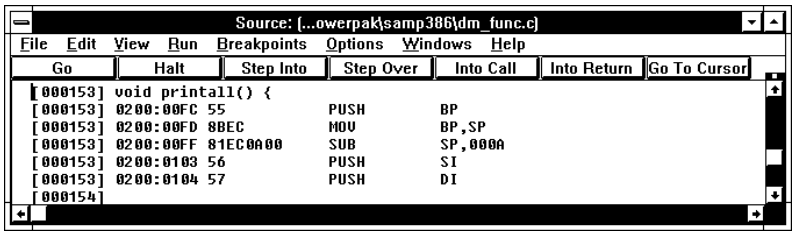
You can move the Source window cursor to the entry point of a function from any occurrance of the function name.

The last search positioned the cursor at the first occurrance of the printall function name in the dm_func module. Double-click on the function name. The Function pop-up menu appears. Choose Go To Source. The Source window displays the first line of the function.

Function pop-up menu accessed by double-clicking on a printall string in the Source window



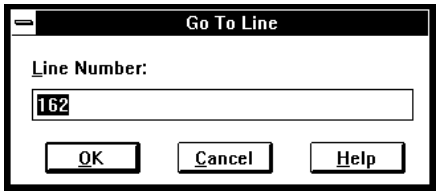
Source window display after choosing Go To Source in the Function: printall pop-up menu



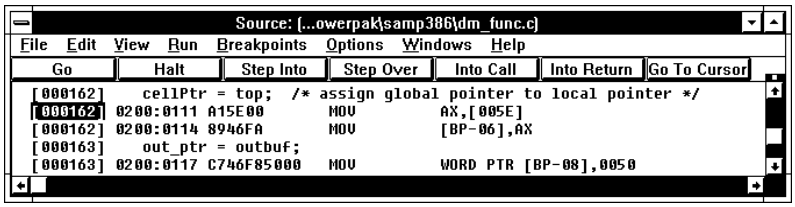
View a Specific Line

You can find any line in the currently displayed module by line number. Open the Edit menu and choose Go To Line. Enter 162 in the Go To Line dialog box.

Go To Line dialog box for finding line 162 in the Source window



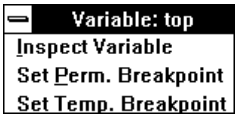
Source window displaying line 162



Inspect a Local Variable

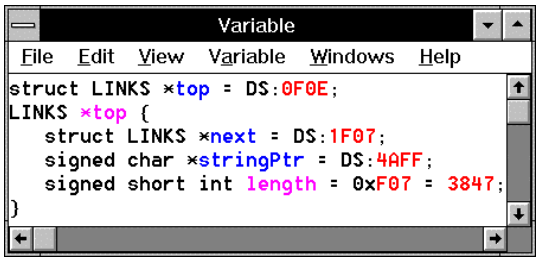
You can examine the definition and value of any variable displayed in the Source window. On line 162, double-click on top. In the Variable pop-up menu, choose Inspect Variable.

Variable pop-up menu accessed by double-clicking on a top string in the Source window



In the Variable window, you can change the values of variables (red) and dereference pointers (blue). Double-click on top, displayed in blue. top points to a structure of two pointers, next and stringPtr, and a short integer, length. Variables out of scope have unknown values.

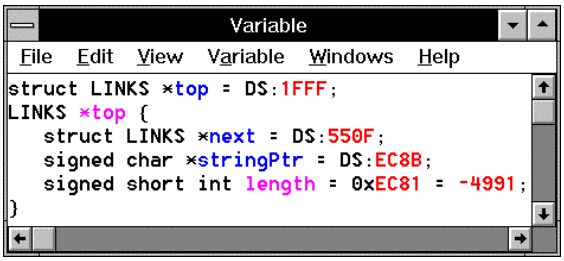
Variable window displayed by choosing Inspect Variable in the Variable: top pop-up menu, with the top pointer dereferenced



Remove the dereferenced structure from the display. Click on the LINKS *top { line; open the Variable menu and choose Delete.

Make top point elsewhere. Double-click on the red 0F0E value of top. In the edit field box appears, enter 1FFF. Since top points to a different location, the values shown for next, stringPtr, and length change.

Result of changing the address in top



Double-click on the 1FFF to open the edit field; enter 0F0E to reset the initially loaded value of top. The displayed values revert.

View the Program Counter Address

Reposition the view and cursor back to the current CS:EIP (program counter). In the Source window, open the Edit menu and choose Go To CS:EIP. The display returns to the startup module.

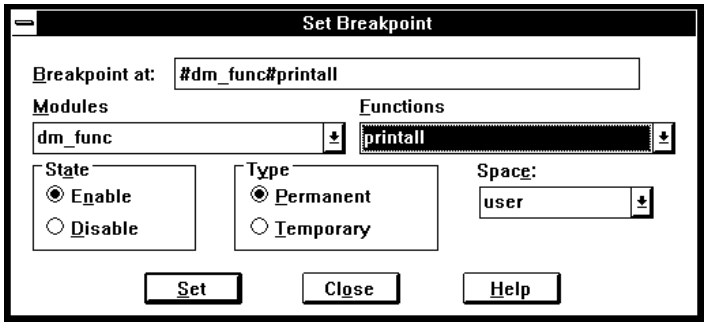
Control Emulation With Breakpoints

In this part of the tutorial, you will learn different ways to run and halt emulation, including the use of breakpoints, the source cursor, and immediate emulation controls.

Set a Breakpoint With the Source Window Breakpoints Menu

In the Source window, open the Breakpoints menu and choose Set Breakpoint. In the Set Breakpoint dialog box, select the dm_func module and the printall function. Choose Set to set the breakpoint.

Set Breakpoint dialog box, setting a permanent, enabled breakpoint at the beginning of the printall function in the dm_func module



The breakpoint is marked with a red highlight on the first assembly line at the specified address. Close the Set Breakpoint dialog box.

Set a Breakpoint With the Source Window Mouse Cursor

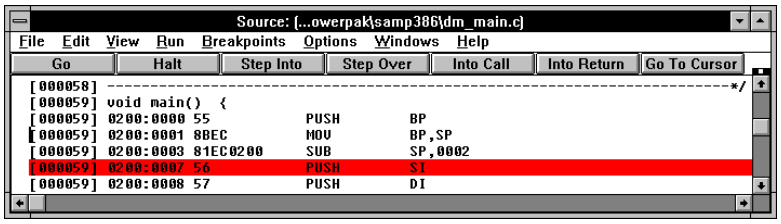
You can use the mouse to set a breakpoint in the Source window. When you point the mouse to the left of a source line, the SLD software displays a cross-hair cursor (shown here in the left margin).

Use the mouse to set a breakpoint at the beginning of main:

1. Browse to the dm_main module. In the Source window, open the File menu; choose Browse Modules; in the Browse Modules dialog box, select dm_main from the list box.
2. Scroll the display to line 59. Open the Edit menu; choose Go To Line; in the Go To Line dialog box, enter 59. The source display in mixed view shows the source line 59 followed by the associated instructions disassembled from memory.
3. Position the mouse pointer in the left margin of the Source window beside the assembly line [000059] 0200:0007 56 PUSH SI. When the mouse pointer becomes a cross-hair cursor, click a mouse

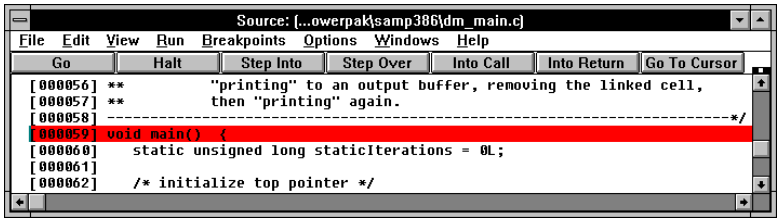
button to set the breakpoint. The primary button sets a permanent breakpoint. The secondary button sets a temporary breakpoint. The breakpoint line is highlighted in red.

Source window showing breakpoint line in mixed source and assembly view



Mixed view highlights the assembly line with the breakpoint. Source-only view highlights the entire source line, regardless of which instruction or statement has the breakpoint. Open the View menu and choose Source Only to display source without interleaved disassembly.

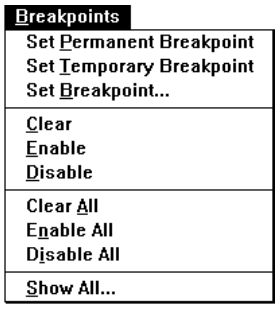
Source window showing breakpoint line in source-only view



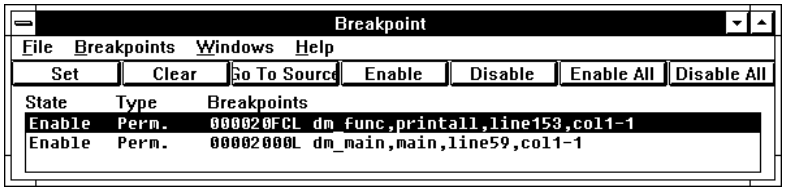
View the Currently Set Breakpoints

Open the Source window Breakpoints menu and choose Show All. The Breakpoint window appears, listing all currently set breakpoints.

Source window Breakpoints menu

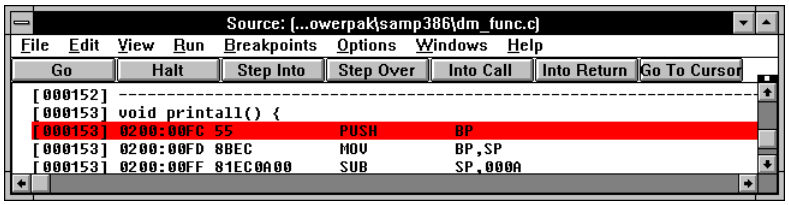


Breakpoints window opened by the Source window Breakpoints menu Show All item

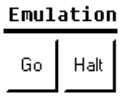


With the highlight on the printall breakpoint, choose the Go To Source button. The Source window displays the breakpoint line, as follows.

Source window display (in mixed source and assembly view) corresponding to a line in the Breakpoint window



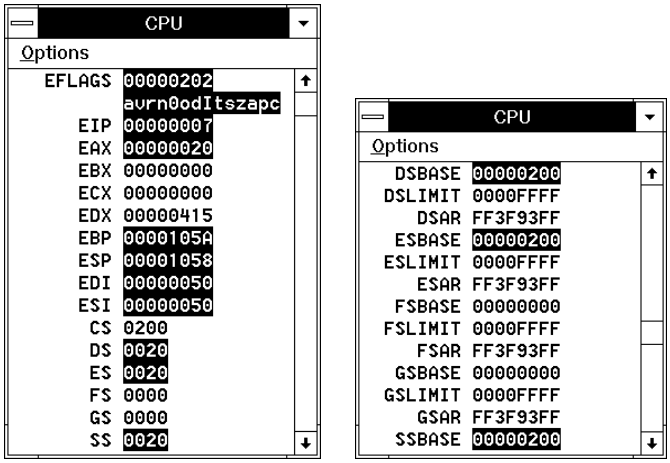
Emulate To a Breakpoint



On the Toolbar, choose the Go button. Emulation starts, then breaks on the first breakpoint. The new CS:EIP address is marked by >> (the program counter) as well as by the red highlight (a breakpoint).

Look at the CPU window. During emulation, some CPU register values changed and are highlighted.

Registers in CPU window modified by program execution



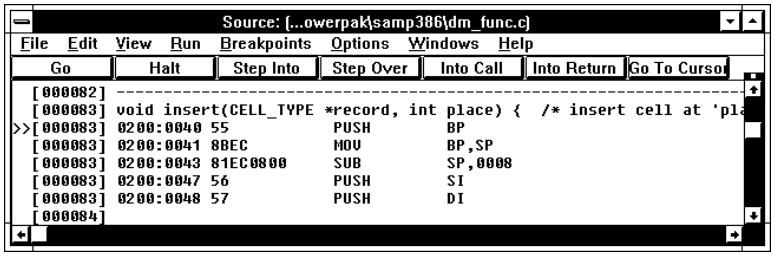
Control Emulation With Buttons and Menus

Emulate By Stepping

In the Source window, open the Run menu and choose Step Into to step into an executable statement. The CS:EIP moves to the next executable statement. In mixed view, stepping is one assembly line at a time; in source-only view, stepping is one source line at a time. Ensure the display is in mixed view before continuing this tutorial.

Continue choosing Step Into until the CS:EIP is on the disassembly line [000067] 0200:0017 E82600 CALL dm_func#83 (insert). The next Step Into will emulate into the insert function call, changing the source display to show the CS:EIP indicator on the first executable line of the insert function. Choose Step Into again.

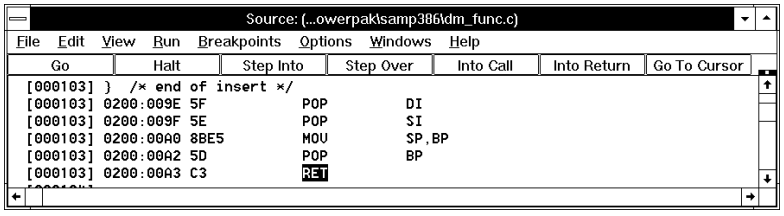
Source window display after stepping into the insert function



Emulate to the Cursor

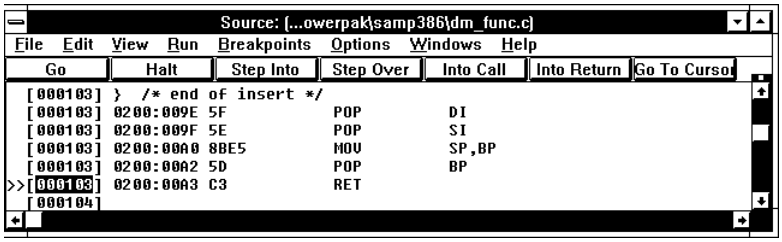
You can progress emulation to a specific line without setting a breakpoint. Open the Source window Edit menu; choose Go To Line and enter 103. With the mouse, click somewhere on the line [000103] 0200:00A3 C3 RET (not in the left margin) to position the source cursor on the return instruction.

Source window cursor on RET instruction



Open the Run menu and choose Goto Cursor. The CS:EIP moves to the location you selected with the cursor, as shown in the following.

Source window displaying current execution point (CS:EIP), indicated by >>

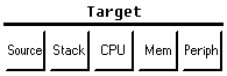


Go To commands in the Edit menu move the cursor without without changing the CS:EIP. Go To commands in the Run menu change both the CS:EIP and the source cursor.

View the Call Stack

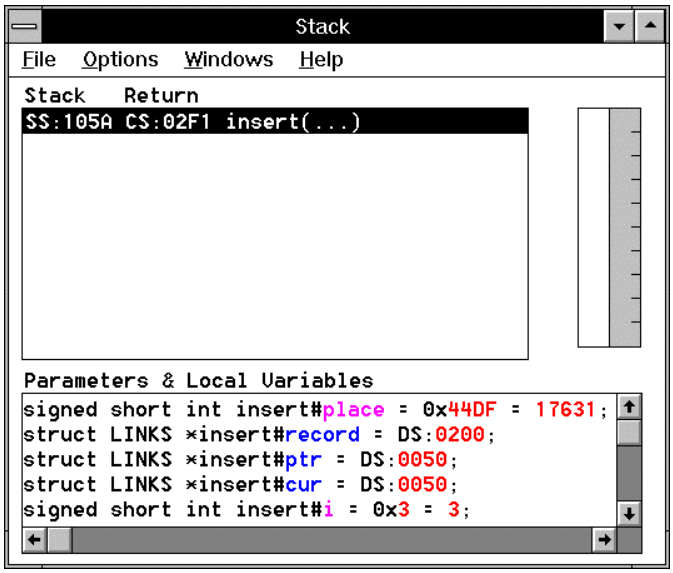
Once you load and execute a program through several function calls, the stack list contains stack frames. In this part of the tutorial, you will learn how to configure and read the Stack window.

Open the Stack Window



On the Toolbar, choose Stack.

Stack window showing the stack and return addresses, parameters, and local variables of the function where emulation is halted



The Stack window contains two panes and a stack meter:

Stack list — (top pane) displays the nested call sequence leading up to the current context. Each call is shown as a single line, called a frame. Click on a frame in the stack list to highlight (select) it. Stack and code addresses appear, depending on the Options menu settings, in each frame:

- Stack address — is the address of the frame in the stack. Dashes mean the compiler generated no stack frame for the function.
- Code address — is the address in the calling function to which the program counter will return.

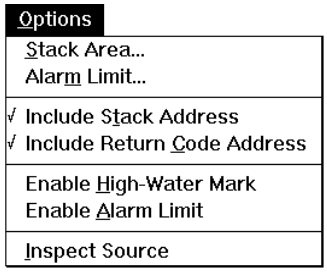
Variables list — (bottom pane) displays the local variables and parameters of the selected frame in the stack list. An

unknown value for a variable means the variable is not in scope and is not saved on the stack. The colors in the Stack window variables list have the same meanings as in the Variable window.

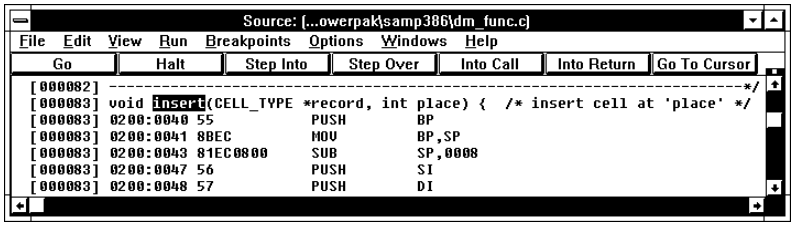
Stack meter — (to the right of the top pane) shows what percentage of the stack area is currently used. Blue shows current stack usage; yellow shows stack underflow; purple shows stack overflow. You can configure the stack meter for statistical information about stack area usage.

You can view any listed function's source. With the highlight on the insert frame in the stack list, open the Options menu and choose Inspect Source.

Stack window Options menu



Source window showing source and disassembly for the insert function on the stack

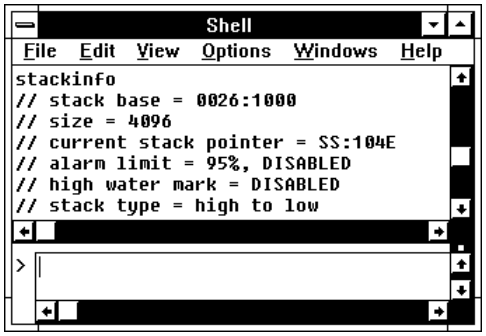


Track Stack Usage Statistically

In this section, you will configure the stack meter to show stack usage statistics. For the stack meter to be active, the stack base and stack size known to the emulator must logically match the current SS and accomodate the current ESP used by the program.

The initial stack base was reported in the Load Complete information box immediately after you loaded demo.omf. This value is unchanged because you have entered no Shell or menu command to change it. To retrieve the stack base, enter stackinfo in the Shell window.

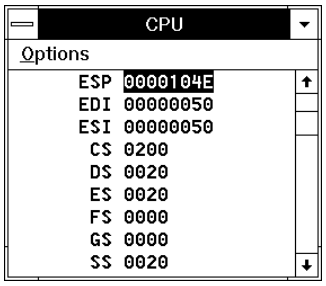
Shell window StackInfo command



The initial stack base was put into the loadfile during linking. The startup code changes SS:ESP.

Source of startup code initializing SS:ESP

CPU window showing the current values of SS and ESP, with ESP highlighted to indicate change during emulation

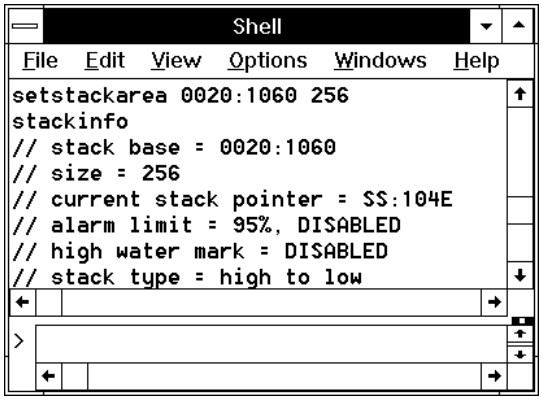


The stack meter in the Stack window monitors stack activity within an area defined by a stack base and size that you can set with Shell commands and Stack window menu items. When the SS:ESP is inside this monitored stack area, the stack meter is active and the statistical options (stack alarm and high-water mark) can be enabled.

The SS:ESP is 0020:104E, outside of the stack area defined by the Stack window stack base (0026:1000) and size (4096). However, this SS:ESP is consistent with the SS:ESP set in startup (no subsequent code has relocated the stack).

Change the Stack window stack base to 0020:1060 to match the startup SS:ESP. To make small stack usages visible on the stack meter, reduce the Stack window stack size to 256. Use setstackarea for both changes, then stackinfo to display the new monitored area.

Shell window SetStackArea and StackInfo commands

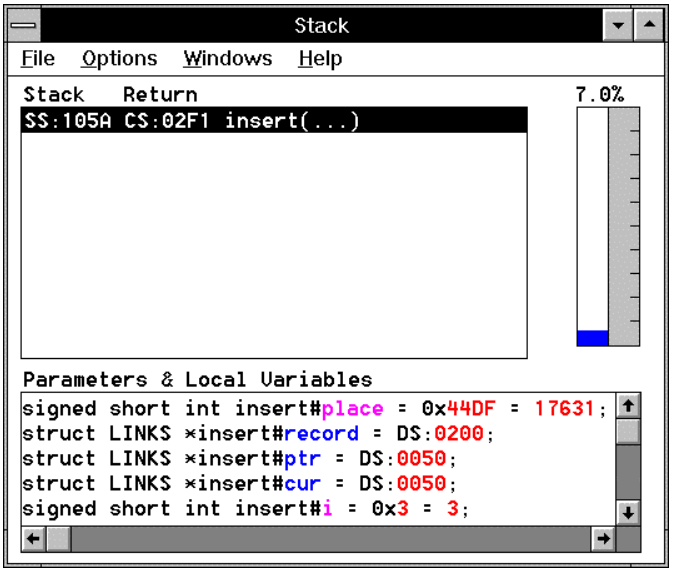


Stack CAUTION

Avoid changing the SS and ESP shown in the CPU window. Changing these registers would affect the stack used by your program.

The stack meter in the Stack window becomes active, showing how much (7%, or about 18 bytes) of the recognized stack area (256 bytes at 0020:1060) is currently in use.

Stack window with active Stack meter

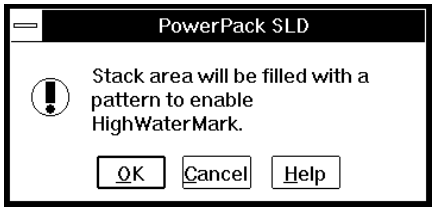


Changing the stack size recognized by the emulator does not affect the amount of memory available to the program for stack activity. Changing the stack base recognized by the emulator does not affect the

SS or ESP. The stack base and size are used only by the emulator to maintain the stack usage statistics.

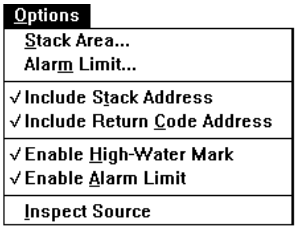
Display the greatest stack usage since stack initialization. Open the Options menu and choose Enable High Water Mark. The high-water mark appears as an arrow on the stack meter (see figure at left). Initially, the high-water mark is set at the current level of stack usage, and any unused part of the monitored stack area is filled with a pattern.

Warning message that a pattern will be written to the unused part of the monitored stack area

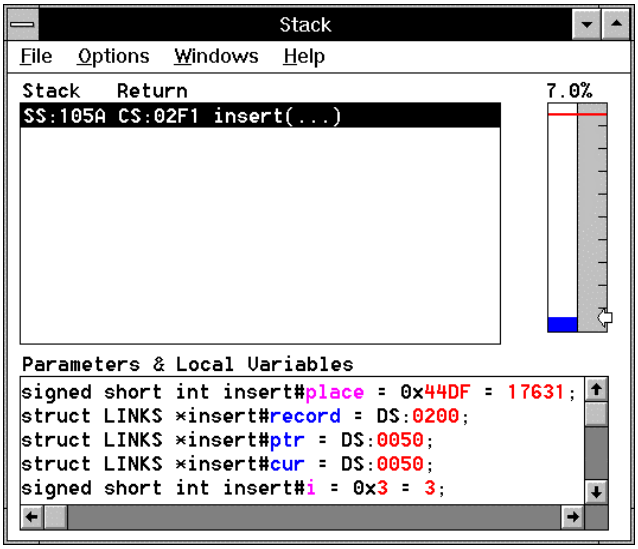


Set a stack alarm to warn when stack usage exceeds a specified percentage of the stack area. Open the Options menu and choose Enable Alarm Limit. The alarm limit appears as a red line at 95% on the stack meter. Each time emulation stops, if the alarm limit is currently exceeded a warning message appears. The following shows the Options menu with the high-water mark and alarm enabled.

Stack window Options menu with high-water mark and alarm limit enabled



Stack meter with high-water mark and alarm limit enabled



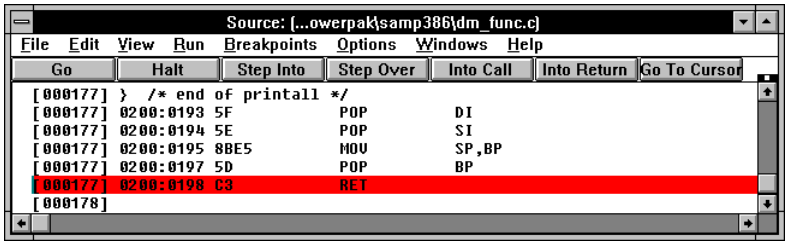
The Stack window is updated only when emulation stops. To monitor the stack during execution, emulate with Step Into/Over Continuously from the Source window Run menu. Each step updates the Stack window. Choose Halt to stop stepping.

Collect and Examine Trace Information

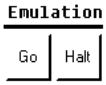
In this part of the tutorial, you will collect and view trace information.

Ensure no breakpoints are set. In the Source window, open the Breakpoints menu and choose Clear All. In mixed view, set a breakpoint on the RET instruction at the end of printall.

Source window showing breakpoint on printall RET instruction

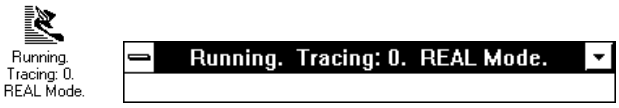


Collect Trace Information



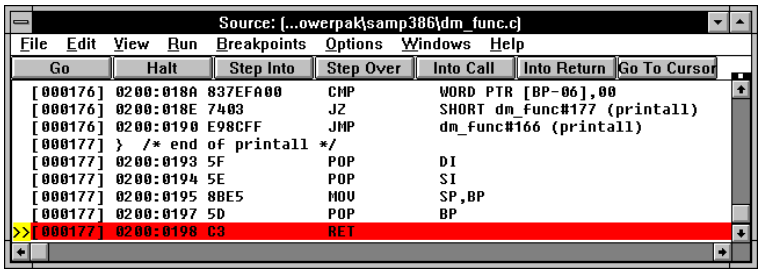
On the Toolbar, choose the Go button. Tracing starts when emulation starts and stops when emulation stops. The Status window title lists the emulation and tracing status.

Status icon and window during emulation and tracing

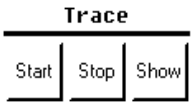


Emulation and tracing stop at the breakpoint.

Source window showing emulation halted at the breakpoint

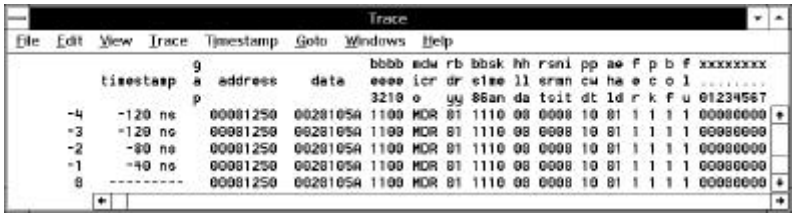


View the Trace as Clock Cycles



On the Toolbar, choose the Show button to display the Trace window. The trace information appears as bus cycles. Scroll back to address 000218A, corresponding to address 0200:018A in the Source window. You can view the opcodes on the data bus corresponding to the opcodes in the Source window disassembly. (In a Memory window showing the hexadecimal values, you can view the same opcodes.) Besides the opcodes at code addresses, other values appear on the data bus associated with addresses not among the loaded code. These are memory reads and writes resulting from the opcode executions.

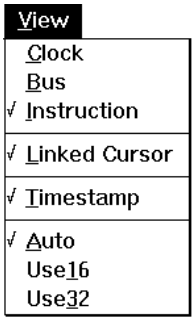
EA-486 trace display



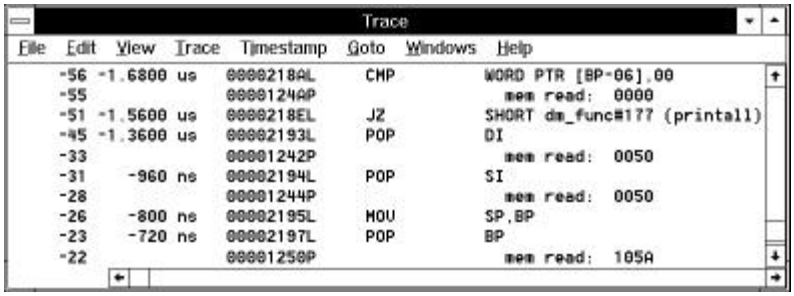
View the Trace as Disassembly

In the Trace window, open the View menu and choose Instruction. The trace information is disassembled.

Trace window View menu specifying trace be displayed as disassembly

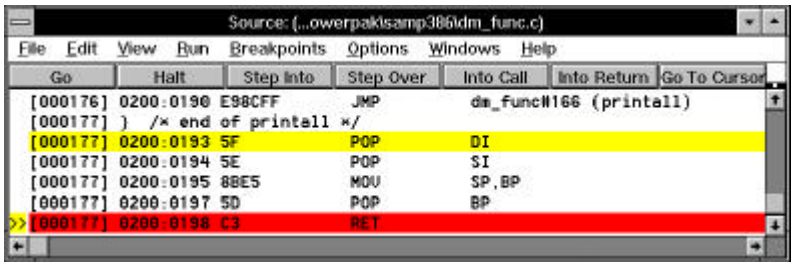


Disassembled trace in Trace window



With the Trace window in instruction view, open the View menu again and choose Linked Cursor. Yellow highlights appear on corresponding lines in the Source and Trace windows. Scroll the Trace window and observe how the Source window scrolls synchronously.

Source window display corresponding to Trace window display (linear address 02193, appearing at frame -45 in the above Trace window)



This is the end of the tutorial for the SW*Plus* emulator.

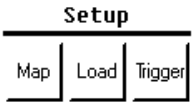
Control Emulation and Tracing With Triggers

The remainder of this tutorial requires an EA emulator.

In this part of the tutorial, you will:

- Define events and triggers for collecting trace information.
- View the collected trace information and find the triggering event.

Configure The Trigger Window

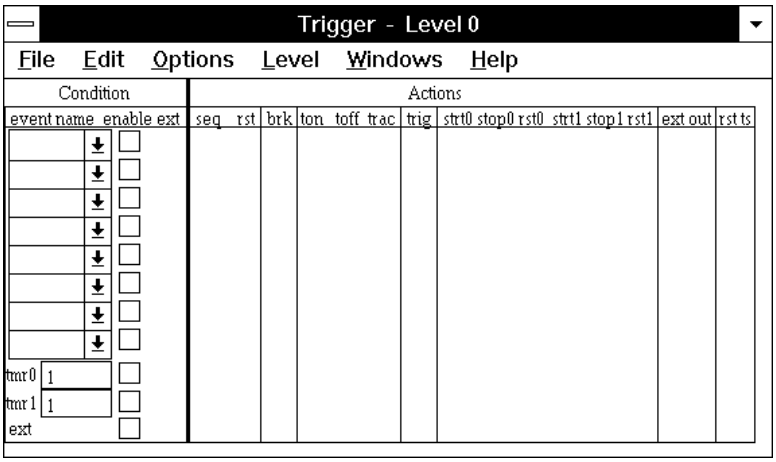


On the Toolbar, choose the Trigger button. The Trigger window has two panes:

Condition	(on the left) causes the trigger. Conditions can be events, counter or timer values, and external active-low signals.
Actions	(on the right) respond to each condition. Actions can control trace collection, emulation, and trigger conditions.

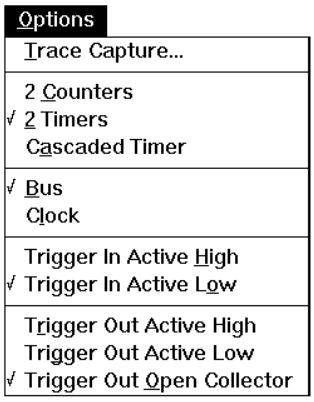
If the labels on the conditions and actions are displayed incorrectly, a screen font manager such as Adobe Type Manager may be overriding the SLD software default font. In Windows, turn off the font manager.

Trigger window with 2 timers enabled



If the Trigger window on your screen shows a pair of counters or timers, open the Options menu and choose Cascaded Timer.

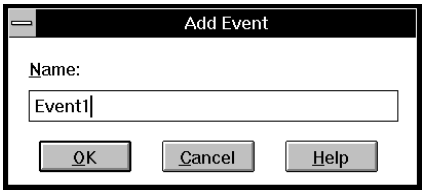
Trigger window Options menu with 2 timers enabled



Define an Event to Trigger Trace Collection

In the Trigger window, click on an event area (or open the Edit menu and choose Events). The Event window appears. Since no events are yet defined, the Add Event dialog box also appears. Enter the name Event1.

Add Event dialog box, defining the new Event1 event.

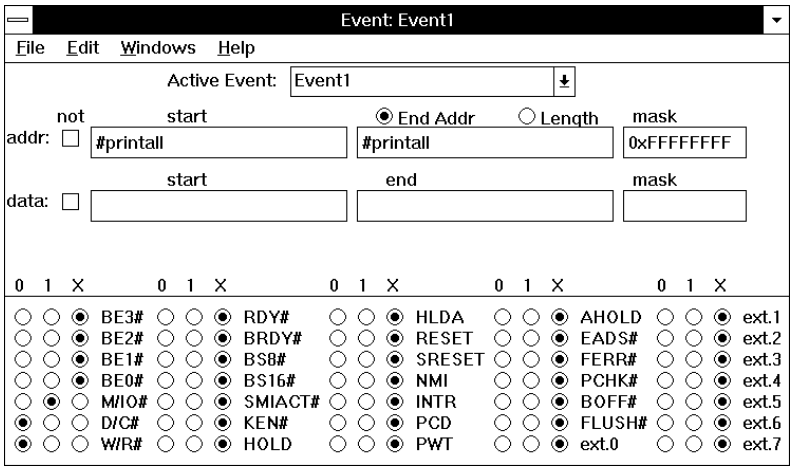


Editing the Event window differs from filling-in a dialog box. Pressing <Enter> has no effect on the field you are editing. Pressing <Delete> can cause an error. To ensure a field accepts an entry:

1. Position the cursor in the field.
2. Type the value.
3. Move the cursor to a different field.

Edit the Event window as shown in the following figure (the signals in the Event window may be different for your processor), defining Event1 as the execution of an instruction from any of the first 10 memory locations where printall is loaded. After filling-in all necessary fields, move the cursor to a different field and close the Event window.

Event window defining Event1 as a memory code read (instruction fetch) at the beginning of the printall function



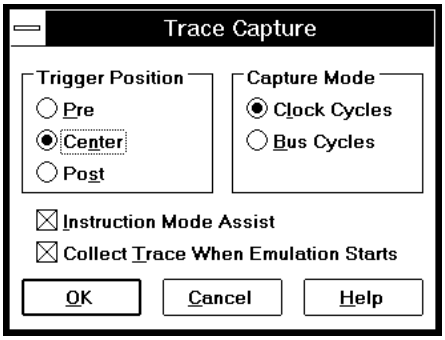
Specify Trace Capture Options

Open the Trigger window Options menu Trace Capture dialog box and select the options as follows:

- Trigger Position Center, to stop trace 125000 cycles after a trig action
- Capture Mode Clock Cycles, to capture trace as clock cycles that can be disassembled
- Instruction Mode Assist, to capture the branch messages needed for disassembly
- Collect Trace When Emulation Starts, to start tracing when you start emulating

Trace Capture dialog box specifying that:

- Trace capture starts when emulation starts and stops 125000 clock cycles after a trig action.
- Clock cycles and branch messages are captured.



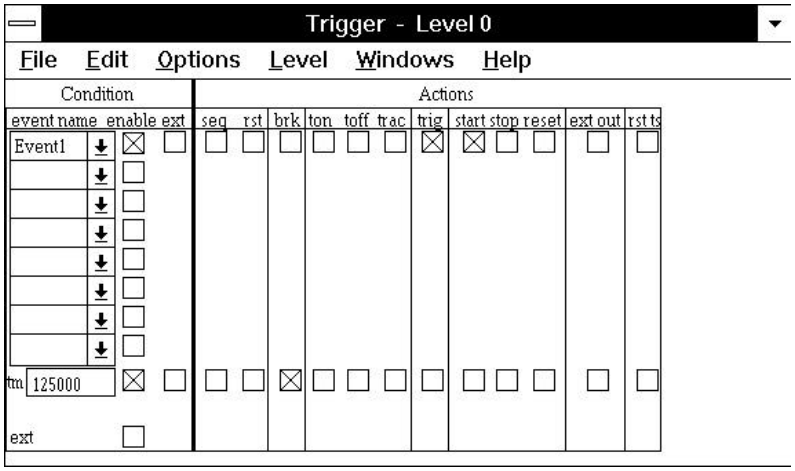
Define the Action to Take When an Event Occurs

In the Trigger window, select the top Event box to specify Event1. Select the enable box to enable the trigger. Select the trig check box, so when Event1 occurs the emulator will:

1. Trigger.
2. Fill the trace buffer, positioning Event1 according to the Trace Capture specifications.
3. Turn off tracing.

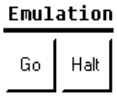
Trigger window (EA), showing the following enabled for the Level 0 trigger:

- the Event1 event condition to perform a trig action and start the timer
- the Cascaded Timer condition to break emulation after 125000 clock cycles



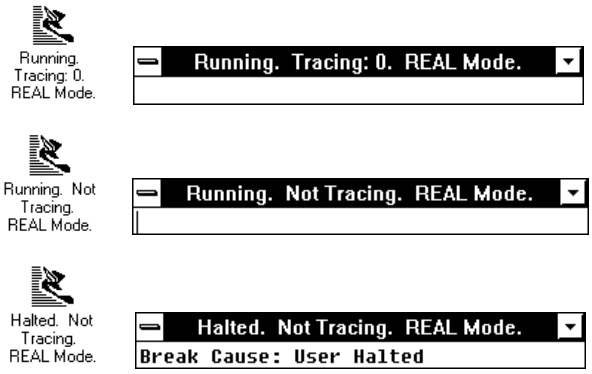
Collect, View, and Save the Trace Information

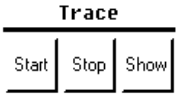
Ensure no breakpoints are set. In the Source window, open the Breakpoints menu and choose Clear All.



From the Emulation section of the Toolbar, choose the Go button. Tracing starts automatically with emulation. When the Event1 trigger occurs, tracing stops and emulation continues. When the Cascaded Timer trigger occurs, emulation halts.

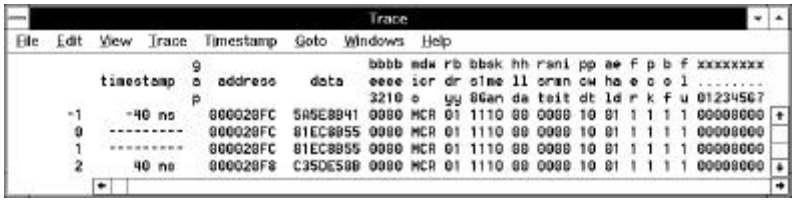
Status window progression during emulation with trace, during emulation without trace, and after emulation halts





From the Trace section of the Toolbar, choose the Show button. The following figure shows trace buffer 0 displayed in a PowerPack Trace window. Read the signal mnemonic labels vertically. The trace frame number appears in the leftmost column.

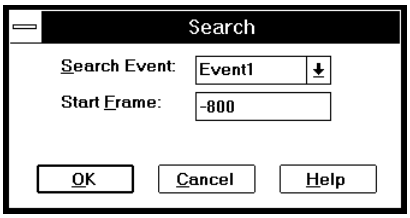
Trace window, displaying trace as clock cycles, with the trigger frame in view



Using the View menu, display the trace as bus cycles and instructions. Observe the frame numbers. Because bus cycles and instructions can span multiple clock cycles, frame numbers in those views are discontinuous.

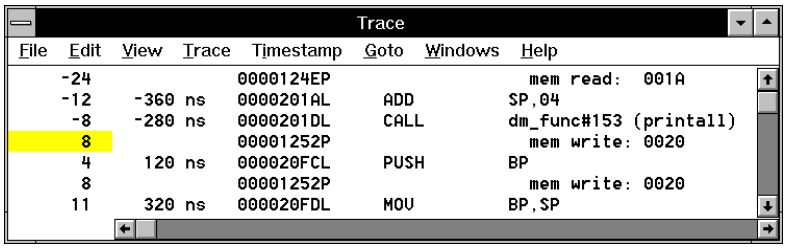
Scroll the section of trace around the trigger frame (frame 0) off screen and find it again with the Edit menu Search dialog box. The trigger frame is the clock cycle matching the condition that caused the trig action: Event1. In the 125000 clock cycles of trace collected after the trigger, the condition may have occured again, so the first occurrance of Event1 is the trigger frame. In the Search dialog box, specify Event1 and a negative frame number.

Search dialog box to find the trigger frame by its event definition



View the trace information as disassembly

Trace window showing trace as disassembly near the trigger frame



Ensure the Source window is open in mixed view. In the Trace window, open the View menu and enable Linked Cursor. The Source window displays the disassembly and source lines corresponding to the Trace window display. Scrolling the Trace window scrolls the Source window synchronously.

Source window showing the in mixed source and assembly corresponding to the Trace window display